CS 453X: Class 5

Jacob Whitehill

Example 2: weather data

- Data from https://www.kaggle.com/selfishgene/historical-hourly-weather-data/data

- Hourly measurements of pressure, humidity, and temperature, and wind speed for different cities in the USA and Israel.

- At each time *t*, how accurately can we predict temperature at time *t*+1hour in Boston?

Linear regression

- Linear regression is one of the few ML algorithms that has an analytical solution:

$$\mathbf{w} = \left(\mathbf{X}\mathbf{X}^\top\right)^{-1}\mathbf{X}\mathbf{y}$$

- **Analytical solution**: there is a closed formula for the answer.

Linear regression

- Alternatively, linear regression can be solved numerically using gradient descent.

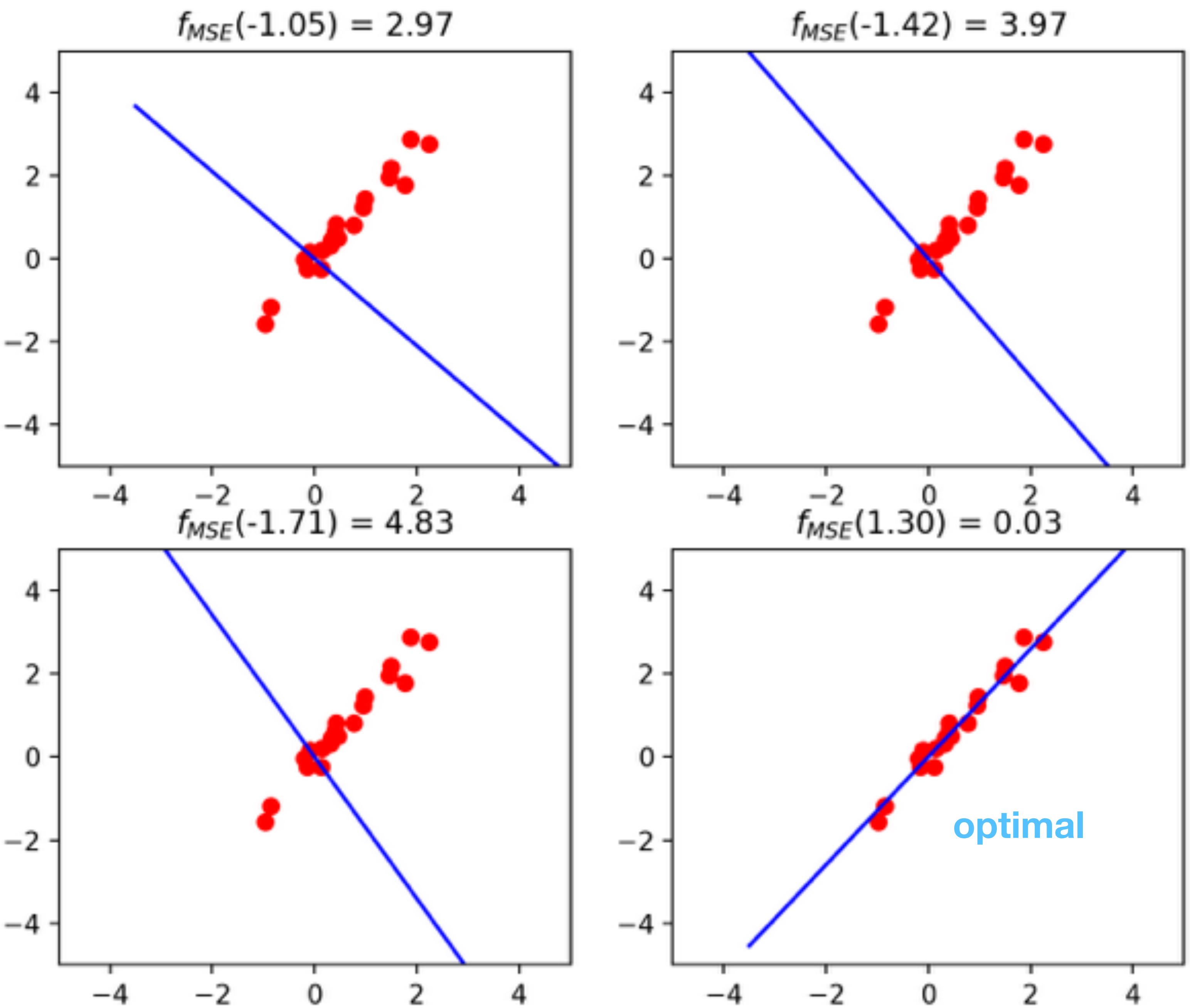
- **Numerical solution**: need to iterate (according to some algorithm) many times to *approximate* the optimal value.

- Gradient descent is more laborious to code than the one-shot solution, but it generalizes to a wide variety of ML models.

Linear regression in 1-d

- Let's look at a simple 1-d example of linear regression again…

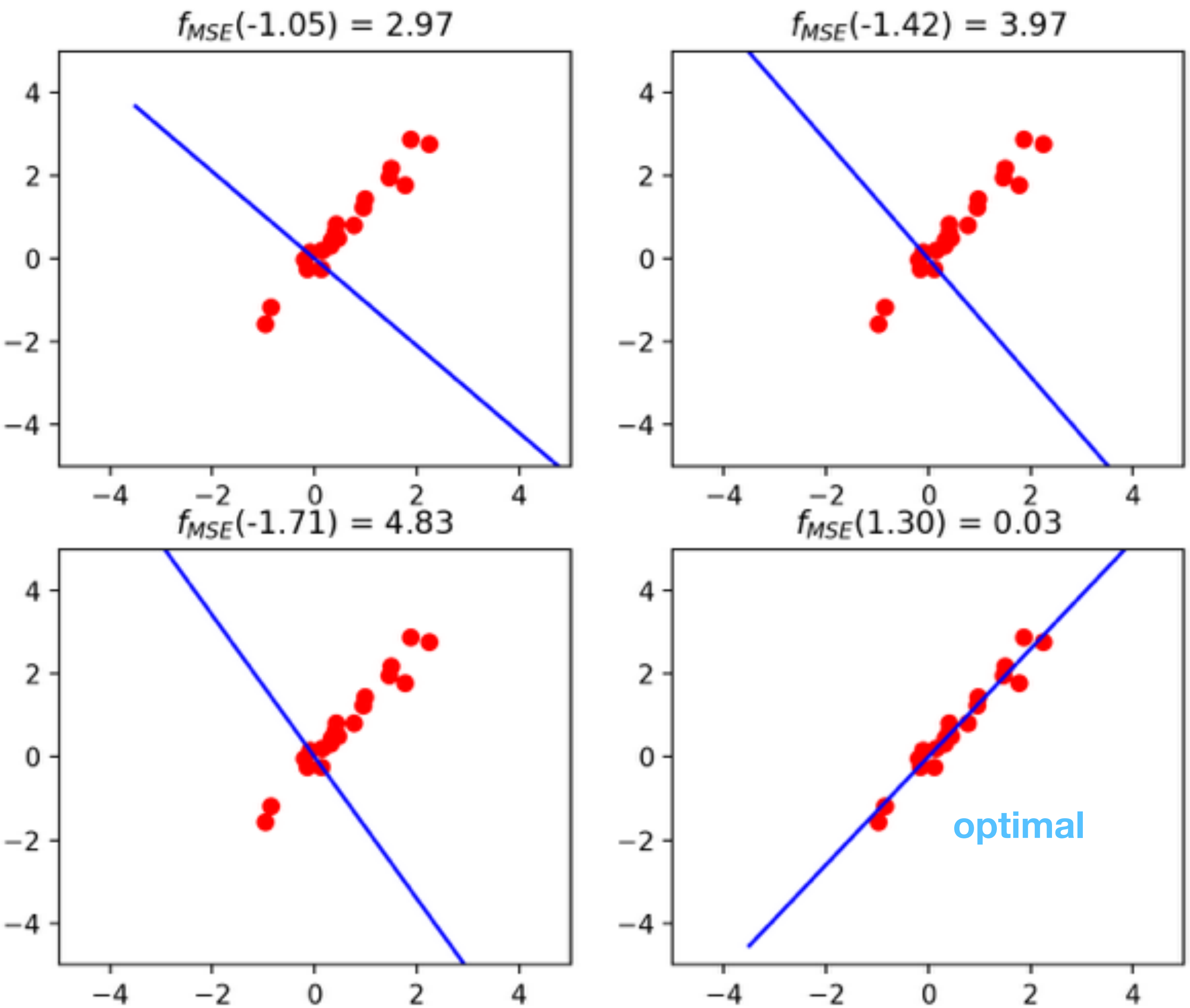
- Here are some different weights **w** and associated costs (f_{MSE}):

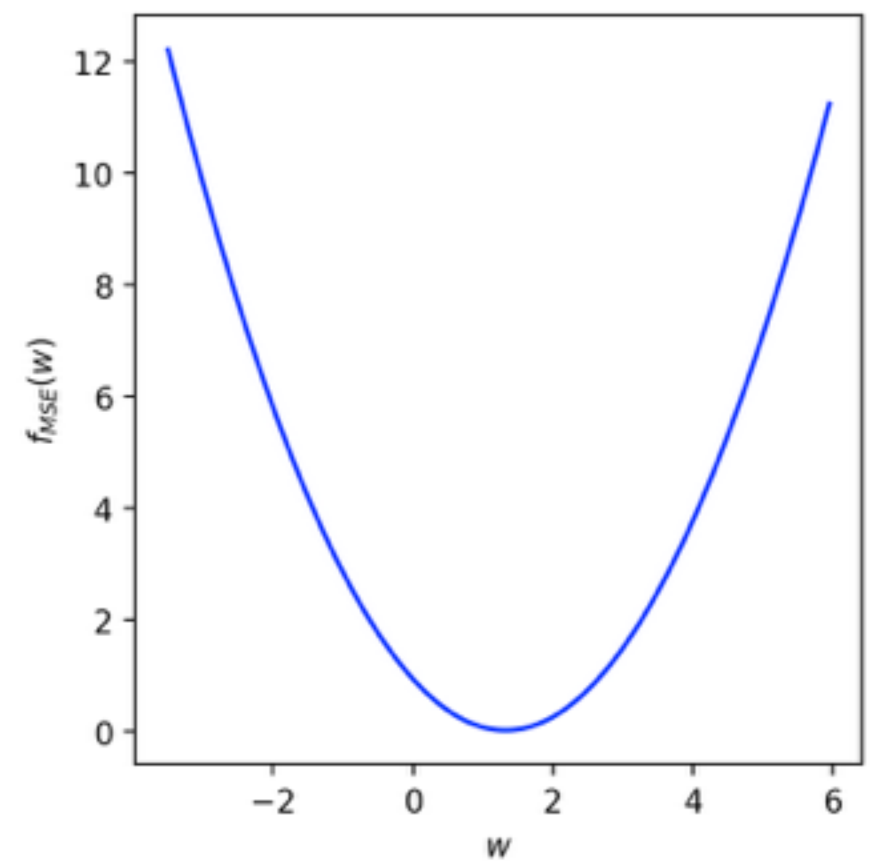


Linear regression in 1-d

- Let’s look at a simple 1-d example of linear regression again…

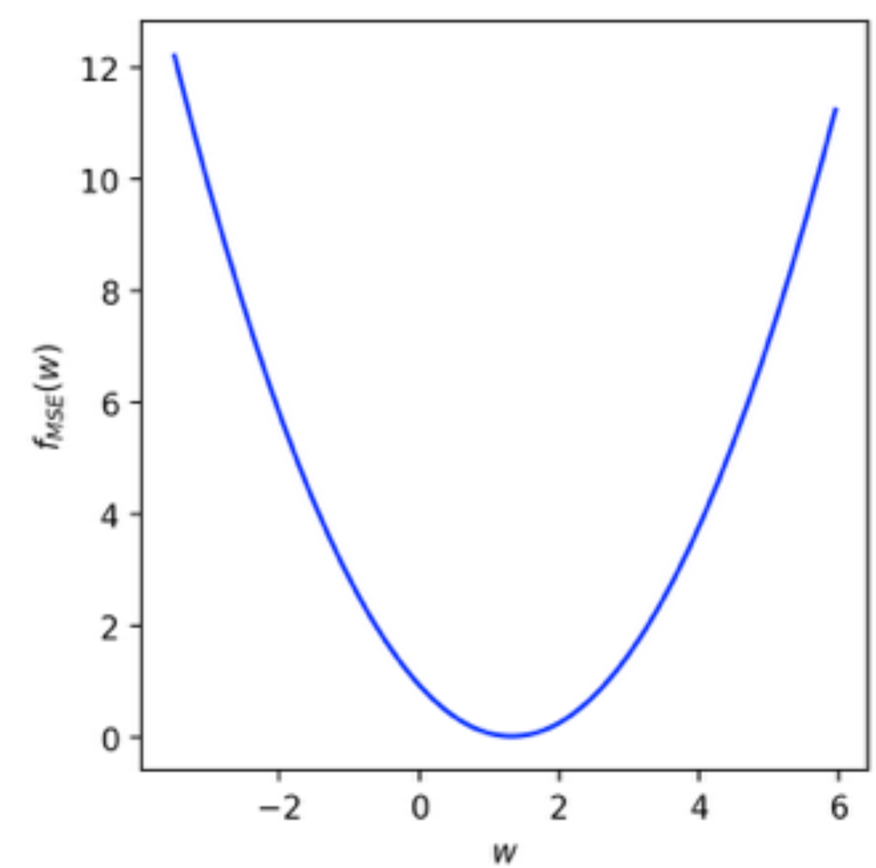
- …and here is the graph of the function $f_{\text{MSE}}(\mathbf{w})$:





Finding the best **w**

- Why not just "jump" to the optimal **w** that minimizes f_{MSE}?



Finding the best w

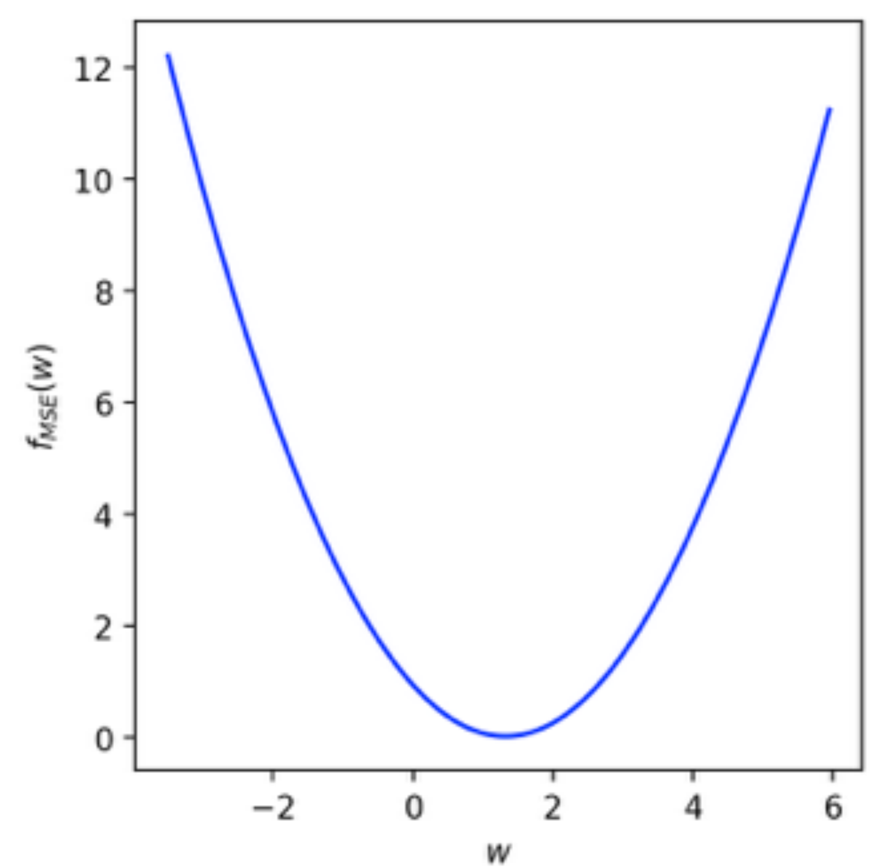
- Why not just “jump” to the optimal **w** that minimizes f_{MSE}?
- In 1-d, we actually could:
  - Just sample many **w** values:

w	-3	-2.99	-2.98	…	5.99	6

  - Compute f_{MSE} for each possible **w**.

$f_{\mathrm{MSE}}(\mathbf{w})$	…	…	…	…	…	…

  - Pick the best one.



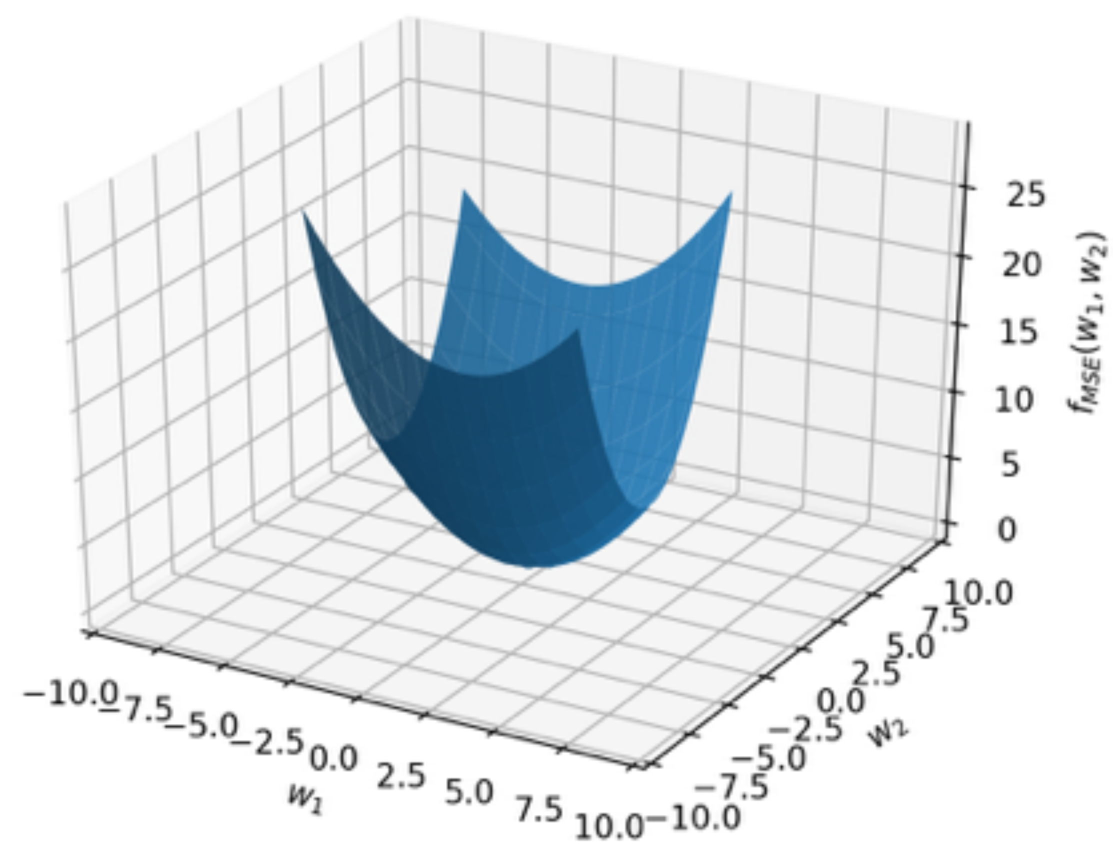
Finding the best **w**

- But what about in higher dimensions (e.g., 2-d).
- We could search through all possible combinations of ($\mathbf{w}_1$, $\mathbf{w}_2$):

$\mathbf{w}_1$

$\mathbf{w}_2$

-3, -3	-3, -2.99	…	-3, 6
-2.99, -3	-2.99, -2.99	…	-2.99, 6
		…	
5.99, -3	5.99, -2.98	…	5.99, 6
6, -3	6, -2.99	…	6, 6



Curse of dimensionality

- As the number of dimensions *m* increases, so does the number of possible values for **w** that we have to probe.
- If we want to sample 100 values per dimension, then we have 100^m values for m dimensions.
- For a 24x24 image, we have *m*=576 dimensions ==> 100^{576}
  - Completely infeasible.

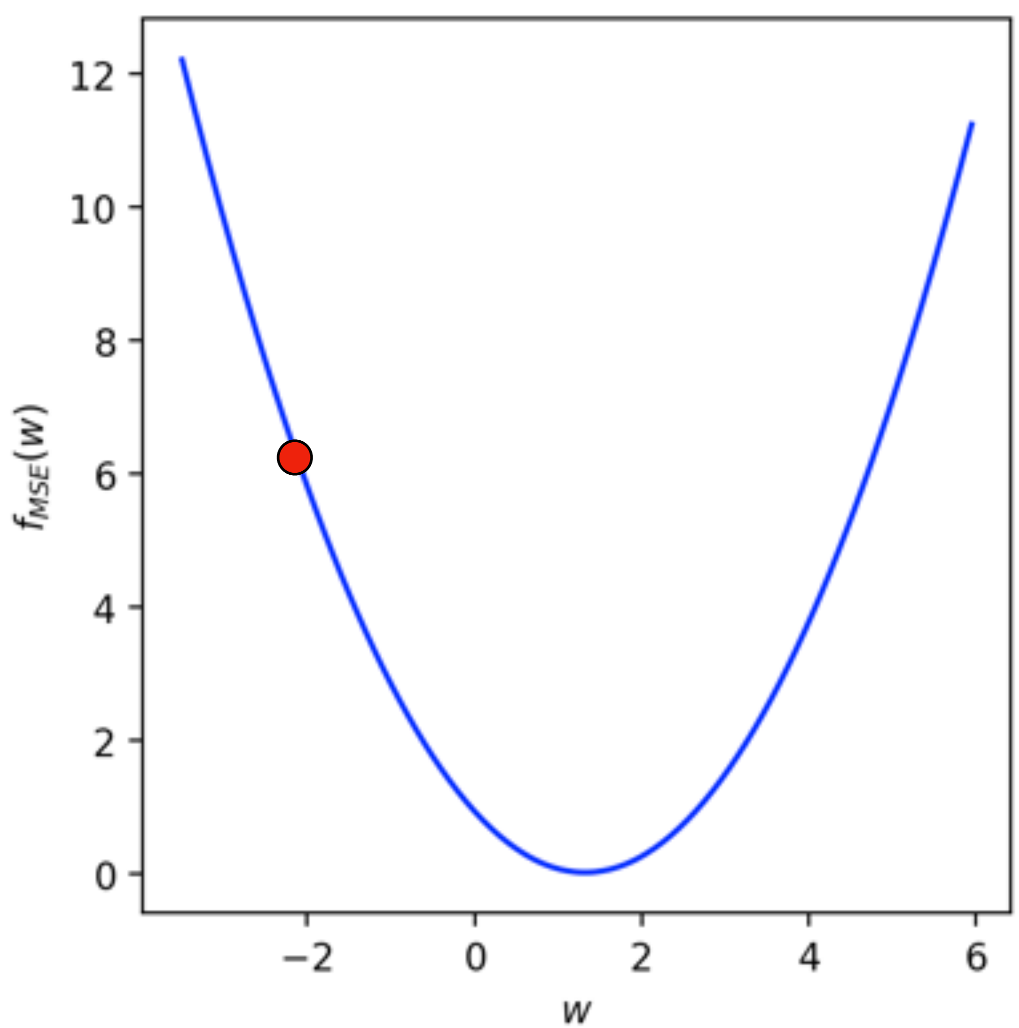
Gradient descent

- Gradient descent is a **hill climbing algorithm** that uses the gradient (aka slope) to decide which way to “move” **w** to reduce the objective function (e.g., f_{MSE}).

Gradient descent

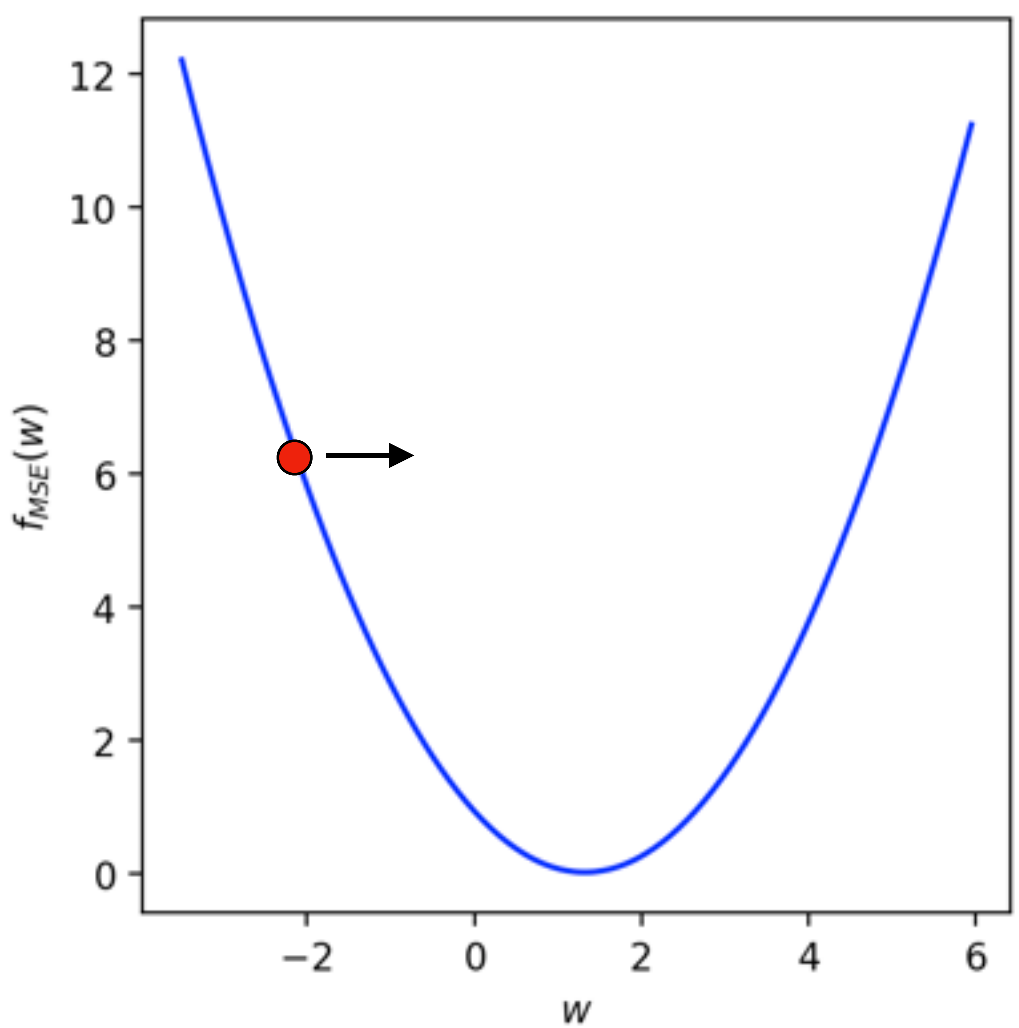
- Suppose we just guess an initial value for **w** (e.g., -2.1).

- How can we make it better — increase it or decrease it?



Gradient descent

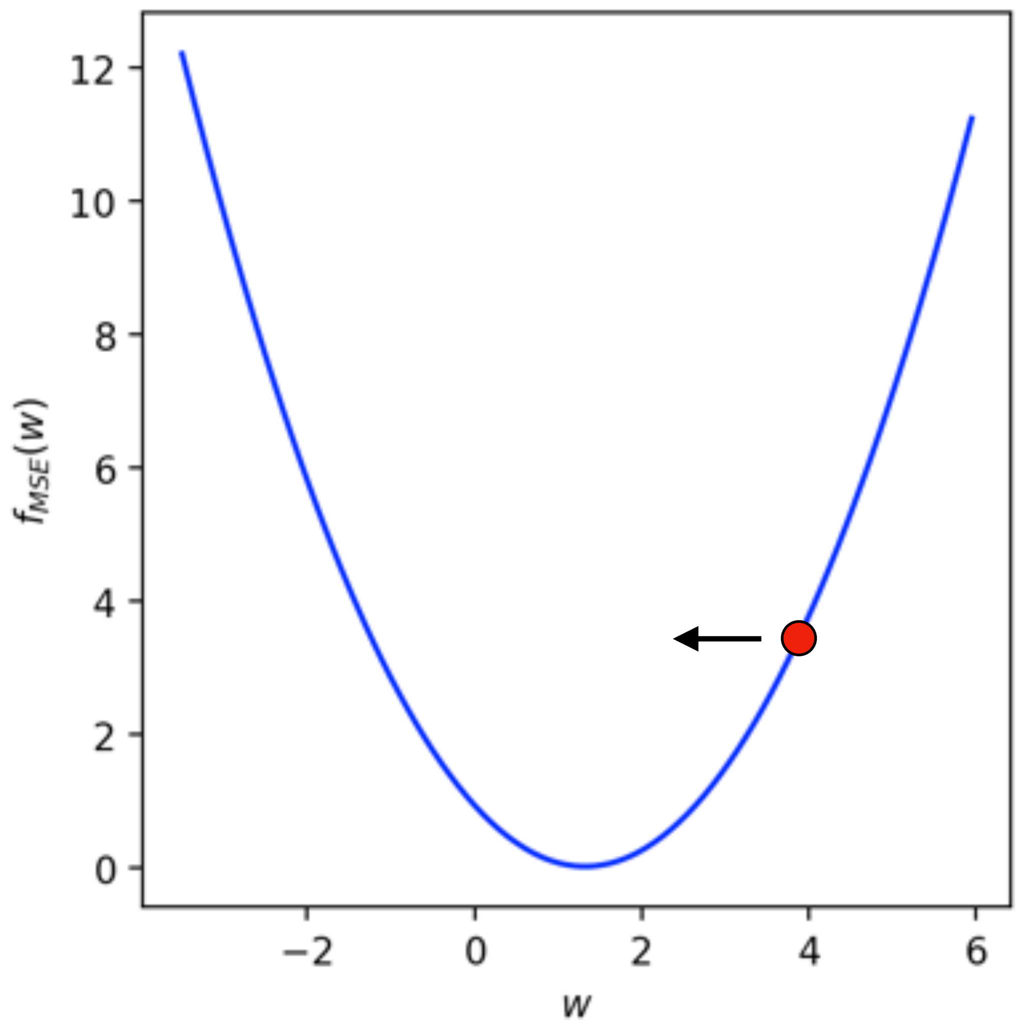
- Suppose we just guess an initial value for **w** (e.g., -2.1).
- How can we make it better — **increase** it or decrease it?
    - What does the **slope** of f_{MSE} tell us to do?



The slope at f_{MSE}(-2.1) is *negative*, i.e., we can *decrease* our cost by *increasing* w.

Gradient descent

- Or maybe our initial guess for **w** was 3.9.
- How can we make it better — increase it or **decrease** it?
  - What does the **slope** of f_{MSE} tell us to do?



The slope at $f_{\text{MSE}}(3.9)$ is *positive*, i.e., we can *decrease* our cost by *decreasing* w.

Gradient descent

- How do we know the slope? Compute the **gradient** of f_{MSE} w.r.t. $\mathbf{w}$:

$$\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[\left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right) \\
&= \frac{1}{n} \mathbf{X} \left(\mathbf{X}^\top \mathbf{w} - \mathbf{y} \right)
\end{aligned}$$

Gradient descent

- How do we know the slope? Compute the **gradient** of f_{MSE} w.r.t. **w**:

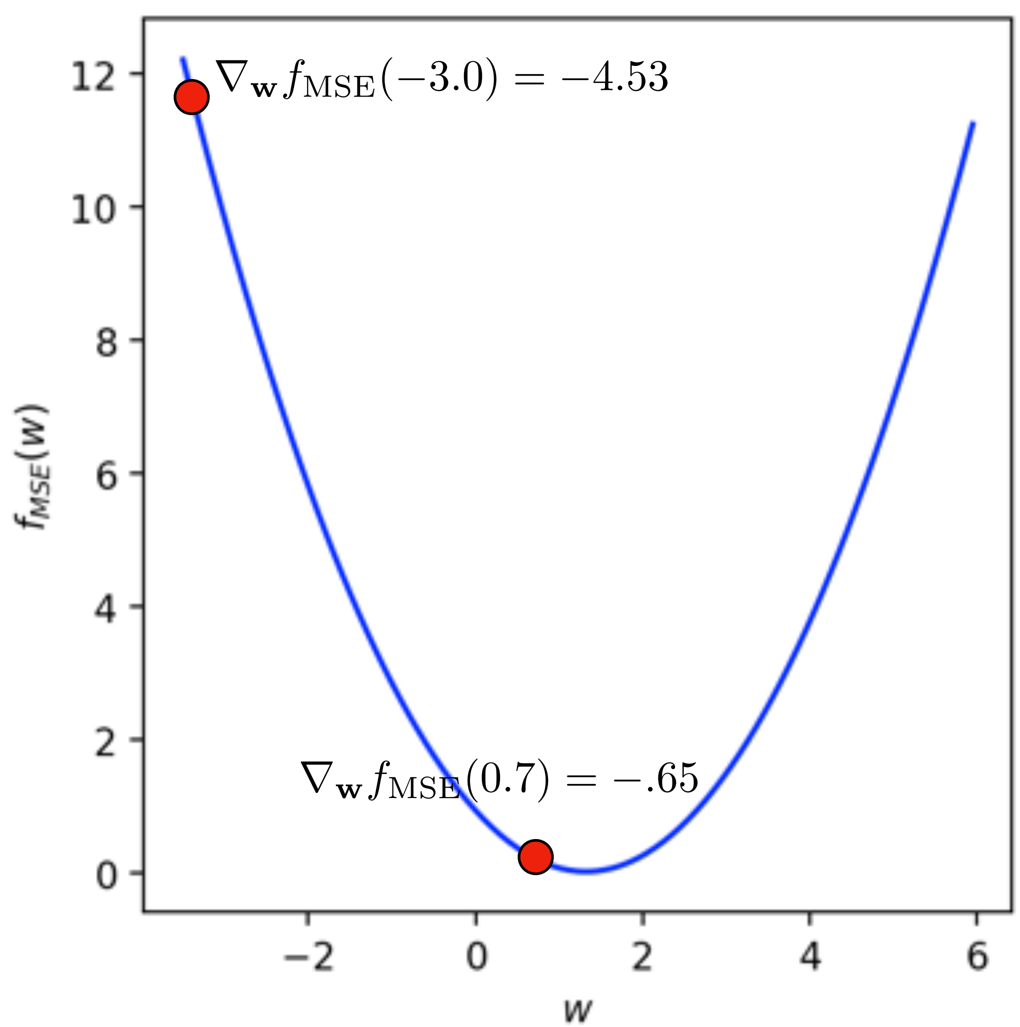
$$
\begin{aligned}
\nabla_{\mathbf{w}} f_{\text{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{2n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left[\left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right)^2 \right] \\
&= \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)\top} \mathbf{w} - y^{(i)} \right) \\
&= \frac{1}{n} \mathbf{X} \left(\mathbf{X}^\top \mathbf{w} - \mathbf{y} \right)
\end{aligned}
$$

- Then plug in the current value of **w**.
(Note that **X** and **y** are computed from the data and are constant.)

Gradient descent

- How *far* do we "move" left or right?

  - Notice that, in the graph below, the **magnitude** of the slope (aka gradient) gives an indication of how far we need to go to reach the optimal **w**.



Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.

Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$

Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update **w** by moving opposite the gradient, multiplied by a **step size** ε. $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$

Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update **w** by moving opposite the gradient, multiplied by a **step size** ε. $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Repeat…

$$\mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(1)})$$

$$\mathbf{w}^{(3)} \leftarrow \mathbf{w}^{(2)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(2)})$$

...

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$$

Gradient descent algorithm

- Set **w** to random values; call this initial choice $\mathbf{w}^{(0)}$.
- Compute the gradient: $\nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Update **w** by moving opposite the gradient, multiplied by a **step size** ε. $\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(0)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(0)})$
- Repeat…

$$\mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(1)})$$

$$\mathbf{w}^{(3)} \leftarrow \mathbf{w}^{(2)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(2)})$$

…

$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \epsilon \nabla_{\mathbf{w}} f(\mathbf{w}^{(t-1)})$$

- …until convergence:

$$|f(\mathbf{w}^{(t-1)}) - f(\mathbf{w}^{(t)})| < \delta$$

δ is a chosen convergence tolerance.

Gradient descent demos

- 1-d
- 2-d